The Microsoft Geospatial Library

Michael Kallay
Microsoft, michka@microsoft.com

Follow this and additional works at: http://docs.lib.purdue.edu/ddad2011

Part of the Geographic Information Sciences Commons


This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
The Microsoft Geospatial Library

Michael Kallay
Microsoft Corporation
DECLARE @p GEOGRAPHY = 'Point(-122 48)'
SELECT *
FROM GeoNames a
WHERE @p.STDistance(a.geog) < 10000
The Library

• Bing for: “Microsoft SQL Server System CLR Types».

• Name: Microsoft.SqlServer.Types.dll
• SQL-style return types: SqlDouble, SqlBool, etc.
  - May be null when undefined.
  - The “real” result in the Value property.

But –

**Does not depend on SQL Server in any way!**

• Documentation on MSDN.
Geospatial Data

Point: Restaurant, gas station

LineString: Road, river

Polygon: State, continent, lake

MultiPoint, MultiLineString, MultiPolygon, GeometryCollection

(The Open Geospatial Consortium object model)
Set Operations

• Union

• Intersection

• Difference

• Symmetric Difference
Spatial Queries

• Overlaps

• Touches

• Contains

• Intersects = any of the above
Numerical Computations

• Length

• Area

• Distance
Other Constructions

• Buffer

• Convex Hull
Other Constructions

- Centroid
- Reduce
Planar vs. Round Earth

Planar – Cartesian x/y coordinates
Round earth – Longitude/latitude coordinates.
Planar vs. Round Earth

Planar – Cartesian x/y coordinates
Round earth: Longitude/latitude coordinates.
But...

North Korea’s missile threat

<table>
<thead>
<tr>
<th>Type</th>
<th>Maximum range</th>
<th>Payload</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodong</td>
<td>1,300 km (810 miles)</td>
<td>700 kg (1,550 pounds)</td>
<td>Currently deployed</td>
</tr>
<tr>
<td>Taepodong-1</td>
<td>Up to 10,000 km</td>
<td>Several hundred kg</td>
<td>Test failed: 1998, not yet operational</td>
</tr>
<tr>
<td>Taepodong-2</td>
<td>10,000-15,000 km</td>
<td>Several hundred kg</td>
<td>Not yet tested</td>
</tr>
</tbody>
</table>

Sources: Task Force for US Korea Policy, Centre for International Policy
Planar vs. Round Earth

Oops, correction...

Flat-earth thinking. Thank you to those readers who pointed out that, by superimposing concentric circles on a Mercator projection, the map in our May 3rd issue greatly underestimated the potential reach of North Korea's missiles. We stand corrected.

North Korea’s missile threat

<table>
<thead>
<tr>
<th>Type</th>
<th>Maximum range</th>
<th>Payload</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodong</td>
<td>1,300 km (810 miles)</td>
<td>700 kg (;550 pounds)</td>
<td>Currently deployed</td>
</tr>
<tr>
<td>Taepodong-1</td>
<td>Up to 10,000 km</td>
<td>Several hundred kg</td>
<td>Test fired 1998, not yet operational</td>
</tr>
<tr>
<td>Taepodong-2</td>
<td>10,000-15,000 km</td>
<td>Several hundred kg</td>
<td>Not yet tested</td>
</tr>
</tbody>
</table>

Source: Task Force for US Korea Policy, Centre for International Policy
Planar vs. Round Earth

What is the distance from Anchorage to Tokyo?
So Why Planar?

A huge amount of data is represented and processed in planar coordinates, e.g. State Plane Coordinate System
Classes

**SqlGeometry**: Planar, complies with the OGC Simple Features spec.

**SqlGeography**: Round earth, closely follows the OGC SFS. (No public round earth spec.)

Very similar syntax.
Subtle differences in semantics.

No inheritance; only two types.
Subtype (e.g. Point, Polygon) is a property.
Hello world Program

using System;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Types;

class Program
{
    static void Main(string[] args)
    {
        SqlGeometry a = SqlGeometry.Parse("POLYGON ((0 0, 10 0, 10 10, 0 0))");
        System.Console.WriteLine(a.ToString());
    }
}

Result:
> POLYGON ((0 0, 10 0, 10 10, 0 0))
Slightly more interesting:

```csharp
using System;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Types;

class Program
{
    static void Main(string[] args)
    {
        SqlGeometry a = SqlGeometry.Parse("POLYGON((0 0, 10 0, 10 10, 0 0))");
        SqlGeometry b = SqlGeometry.Parse("LINESTRING(5 0, 15 10)");
        SqlGeometry c = a.STUnion(b);
        System.Console.WriteLine(c.ToString());
    }
}

> GEOMETRYCOLLECTION (POLYGON(0 0, 10 0, 10 10, 0 0), LINESTRING(10 5, 15 10))
```
The Sink Interfaces

“Don’t call me, I’ll call you…”

```java
public interface IGeometrySink (or IGeographySink) {
    void SetSrid(int srid);
    void BeginGeometry(OpenGisGeometryType type);
    void BeginFigure(double x, double y, double? z, double? m);
    void AddLine(double x, double y, double? z, double? m);
    void EndFigure();
    void EndGeometry();
}
```

SqlGeometry and Sqlgeography methods:

```java
void Populate(IGeometrySink sink);
void Populate(IGeometrySink sink);
```
The Builder Classes

```java
public class SqlGeometryBuilder : IGeometrySink
    Public SqlGeometry ConstructedGeometry // property

public class SqlGeographyBuilder : IGeographySink
    Public SqlGeography ConstructedGeography // property
```
Example: Normalize Longitude

```csharp
public class LongitudeNormalizer : IGeographySink
{
    private SqlGeographyBuilder Builder;

    public LongitudeNormalizer()
    {
        Builder = new SqlGeographyBuilder();
    }

    private double GetNormalized(double longitude)
    {
        if (longitude < -180)
            return (longitude - 180) % 360 + 180;
        else if (longitude > 180)
            return (longitude + 180) % 360 - 180;
        else
            return longitude;
    }
}
```
Example: Normalize Longitude

```csharp
public void BeginGeography(OpenGisGeographyType type)
{
    Builder.BeginGeography(type);
}

public void EndFigure()
{
    Builder.EndFigure();
}

public void EndGeography()
{
    Builder.EndGeography();
}

public void SetSrid(int srid)
{
    Builder.SetSrid(srid);
}
```
Example: Normalize Longitude

```csharp
public void BeginFigure(double latitude, double longitude)
{
    Builder.BeginFigure(latitude, GetNormalized(longitude));
}

public void BeginFigure(double latitude, double longitude)
{
    Builder.BeginFigure(latitude, GetNormalized(longitude));
}
```
Example: Normalize Longitude

```csharp
public SqlGeography Result
{
    get
    {
        return Builder.ConstructedGeography;
    }
}
} // end of class definition
```
Example: Normalize Longitude

```csharp
public SqlGeography NormalizeLongitude(SqlGeography geography)
{
    LongitudeNormalizer normalizer = new LongitudeNormalizer()
        geography.Populate(normalizer);

    return normalizer.Result;
}
```
Under the Hood

Planar
Ambiguous Geometric Queries
Ambiguous Geometric Queries
Ambiguous Geometric Queries
Ambiguous Geometric Queries

Boil down to the sign of rational expressions.

Most of the time resolved with interval arithmetic.

Fallback: Exact arithmetic with big numbers.
The Hemisphere Restriction

• SqlGeography instance cannot exceed a logical hemisphere.
• Construction scene cannot exceed a hemisphere.
What’s New?

• Hemisphere restriction removed!
• Circular arcs.
• Performance improvements.
• Accuracy improvements.
• Aggregates (union, envelope, etc.).
Resources

The library:
- Download from SQL Server 2008 Feature Pack

Documentation:
- Beginning Spatial with SQL Server 2008 by Alastair Aitchison
- Bing for microsoft.sqlserver.types in MSDN.
- Blogs

Serialization format:
- Bing for Microsoft SQL Server CLR Types Serialization Formats.

Sample code: